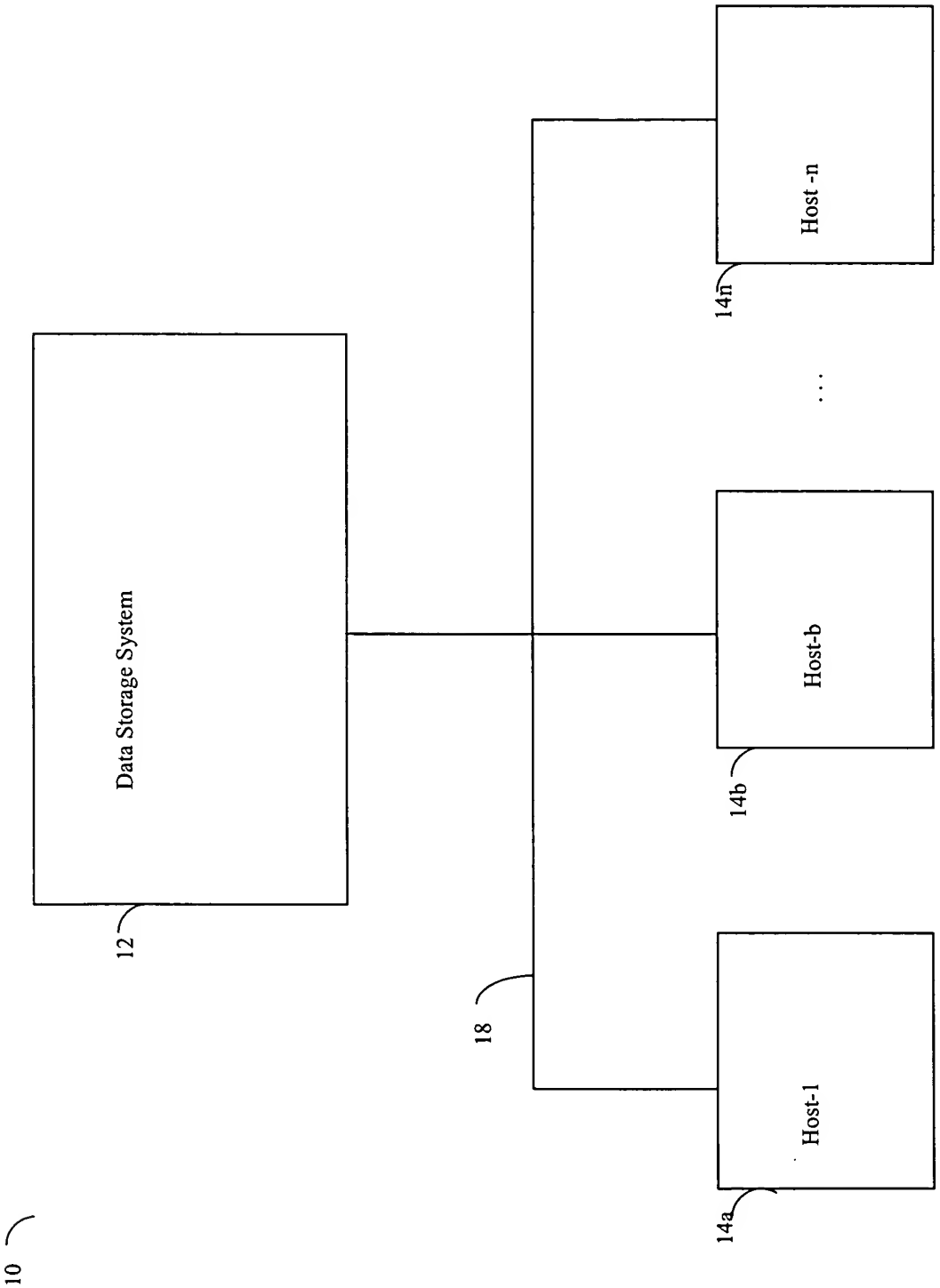


FIGURE 1

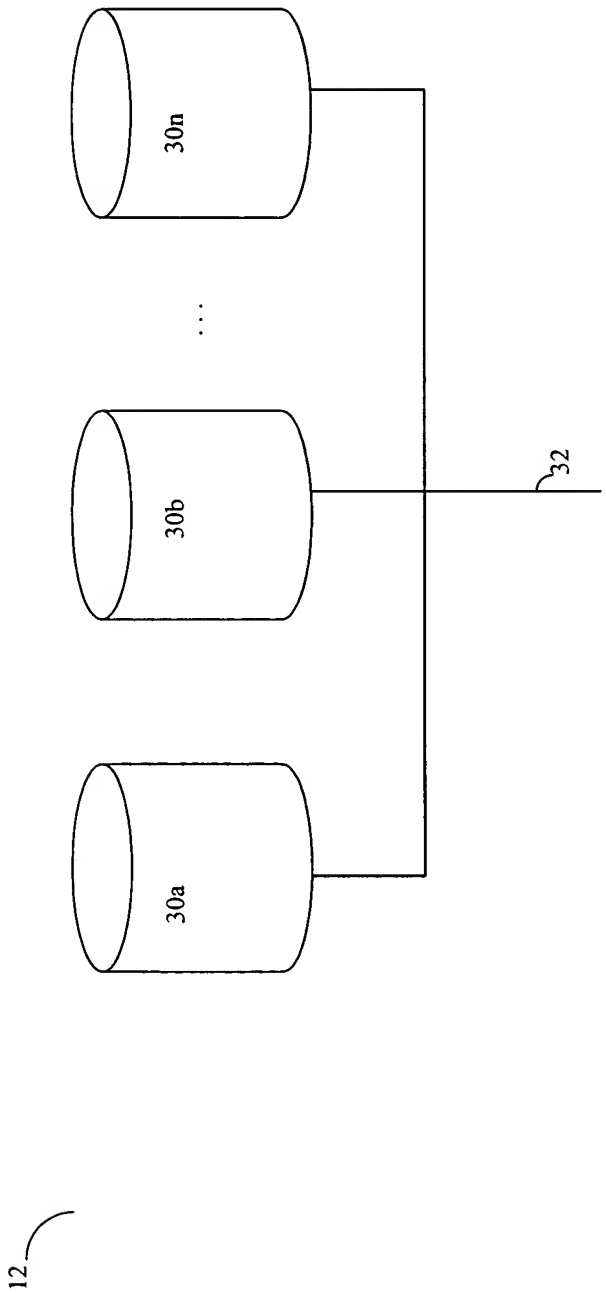


FIGURE 2

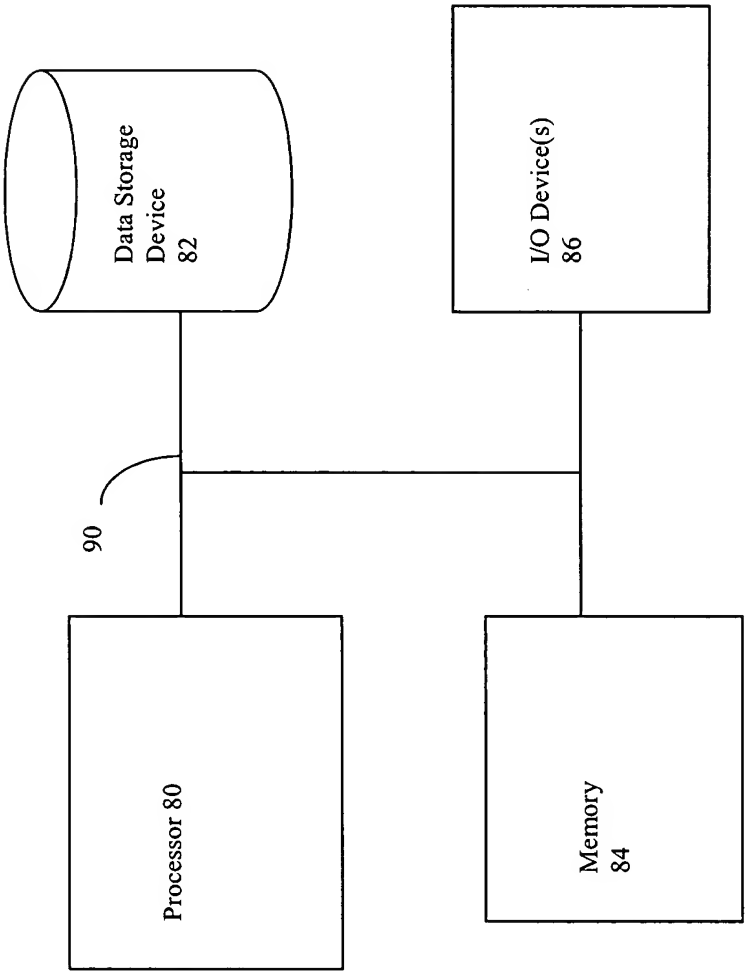

Data Storage Device 82
I/O Device(s) 86
90
Processor 80
Memory 84
14a

FIGURE 3

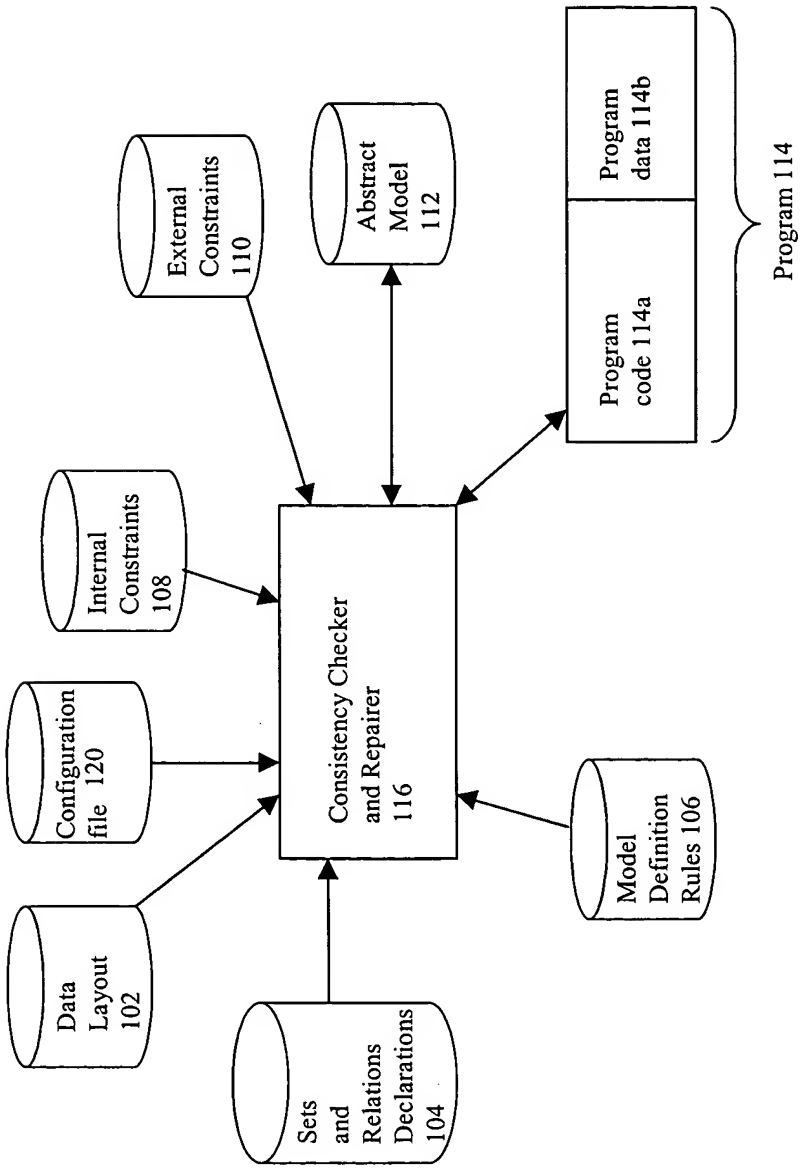

Program data 114b
Program code 114a
Program 114
External Constraints 110
Abstract Model 112
Internal Constraints 108
Configuration file 120
Consistency Checker and Repairer 116
Model Definition Rules 106
Data Layout 102
Sets and Relations Declarations 104
100

FIGURE 4

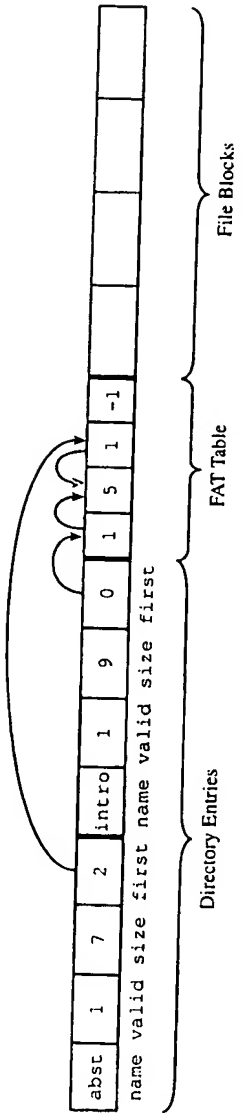


FIGURE 5 Inconsistent File System

120

122

structdefn ::= `struct` *structurename*
(*subtypes structurename*) { *fielddefn** }

fielddefn ::= *type field*; | `reserved` *type*; |
type field[*E*]; |
`reserved` *type*[*E*];

type ::= `boolean` | `byte` | `short` | `int` | *structurename* |
structurename *

E ::= *V* | *number* | *string* | *E.field* |
E.field[*E*] | *E* − *E* | *E* + *E* | *E*/*E* | *E* * *E*

FIGURE 6 Structure Definition Language

124

```
struct Entry {
  byte name[Length];
  byte valid;
  int  size;
  int  first;
}
struct Block { data byte[BlockSize]; }
struct Disk {
  Entry table[NumEntries];
  int FAT[NumBlocks];
  Block block[NumBlocks];
}
```

FIGURE 7 . Structure Declarations

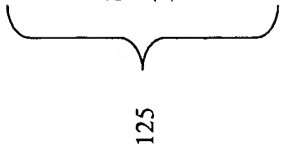


FIGURE 8A

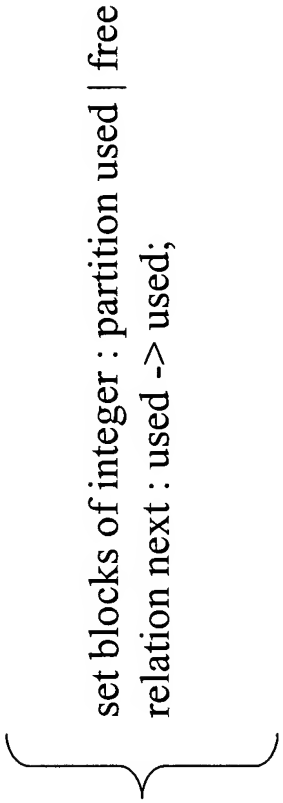


FIGURE 8B

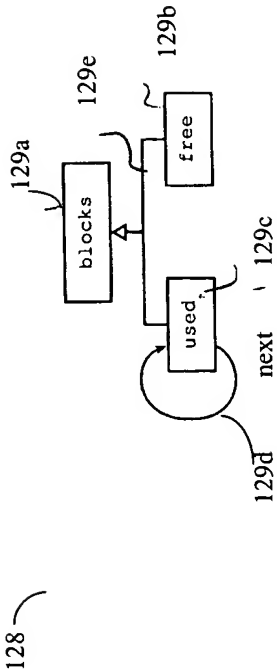


FIGURE 9 Graphical Representation of Object and Relation Declarations

130

$$
\begin{aligned}
C &:= Q, C \mid G \Rightarrow I \\
Q &:= \texttt{for } V \texttt{ in } S \mid \texttt{for } \langle V, V \rangle \texttt{ in } R \mid \\
&\quad\ \texttt{for } V = E \texttt{ .. } E \\
G &:= G \texttt{ and } G \mid G \texttt{ or } G \mid !G \mid E = E \mid E < E \mid \texttt{true} \mid \\
&\quad\ (G) \mid E \texttt{ in } S \mid \langle E, E \rangle \texttt{ in } R \\
I &:= E \texttt{ in } S \mid \langle E, E \rangle \texttt{ in } R \\
E &:= V \mid number \mid string \mid E.field \mid \\
&\quad\ E.field[E] \mid E - E \mid E + E \mid E/E \mid E * E
\end{aligned}
$$

FIGURE 10 **Model Definition Language**

$$
\begin{array}{rcl}
hv & \in & HeapValue = Bit \cup Byte \cup Short \cup Integer \cup Struct \\
h & \in & Heap = \mathcal{P}(Object \times Field \times HeapValue \cup \\
 & & Object \times Field \times \mathbb{N} \times HeapValue) \\
v & \in & Value = \mathbb{Z} \cup Boolean \cup string \cup Struct \\
l & \in & Local = Var \rightharpoonup Value \\
s & \in & Store = Value \times Value \cup Value \\
m & \in & Model = \mathcal{P}(Var \times Store) \\
\mathcal{R} & : & C \rightarrow Heap \rightarrow Local \rightarrow Model \rightarrow Model \\
\mathcal{E} & : & E \rightarrow Heap \rightarrow Local \rightarrow Model \rightarrow Value \\
\mathcal{G} & : & C \rightarrow Heap \rightarrow Local \rightarrow Model \rightarrow Boolean \\
\mathcal{I} & : & I \rightarrow Heap \rightarrow Local \rightarrow Model \rightarrow Model
\end{array}
$$

$$
\begin{array}{l}
\mathcal{R}[\texttt{for } V \texttt{ in } S, C]\ h\ l\ m = \bigcup_{v \in m(S)} \mathcal{R}[C]\ h\ l[V \mapsto v]\ m \\
\mathcal{R}[\texttt{for } \langle V_1, V_2 \rangle \texttt{ in } R, C]\ h\ l\ m = \bigcup_{\langle v_1, v_2 \rangle \in m(R)} \\
\qquad \mathcal{R}[C]\ h\ l[V_1 \mapsto v_1][V_2 \mapsto v_2]\ m \\
\mathcal{R}[\texttt{for } V = E_1 \,..\, E_2, C]\ h\ l\ m = \\
\qquad \bigcup_{i=\mathcal{E}[E_1]\ h\ l\ m}^{\mathcal{E}[E_2]\ h\ l\ m} \mathcal{R}[C]\ h\ l[V \mapsto i]\ m \\
\mathcal{R}[C \Rightarrow I]\ h\ l\ m = \text{if } (\mathcal{G}[G]\ h\ l\ m) \text{ then } (\mathcal{I}[I]\ h\ l\ m) \text{ else } m \\
\mathcal{G}[G_1 \texttt{ and } G_2]\ h\ l\ m = (\mathcal{G}[G_1]\ h\ l\ m) \wedge (\mathcal{G}[G_2]\ h\ l\ m) \\
\mathcal{G}[G_1 \texttt{ or } G_2]\ h\ l\ m = (\mathcal{G}[G_1]\ h\ l\ m) \vee (\mathcal{G}[G_2]\ h\ l\ m) \\
\mathcal{G}[\texttt{!}G]\ h\ l\ m = \neg(\mathcal{G}[G]\ h\ l\ m) \\
\mathcal{G}[E_1 = E_2]\ h\ l\ m = (\mathcal{E}[E_1]\ h\ l\ m) == (\mathcal{E}[E_2]\ h\ l\ m) \\
\mathcal{G}[E_1 < E_2]\ h\ l\ m = (\mathcal{E}[E_1]\ h\ l\ m) < (\mathcal{E}[E_2]\ h\ l\ m) \\
\mathcal{G}[\texttt{true}]\ h\ l\ m = true \\
\mathcal{G}[E \texttt{ in } S]\ h\ l\ m = \langle S, \mathcal{E}[E]\ h\ l\ m \rangle \in m \\
\mathcal{G}[\langle E_1, E_2 \rangle \texttt{ in } R]\ h\ l\ m = \langle R, \langle \mathcal{E}[E_1]\ h\ l\ m, \mathcal{E}[E_2]\ h\ l\ m \rangle \rangle \in m \\
\mathcal{I}[E \texttt{ in } S]\ h\ l\ m = m \cup \langle S, \mathcal{E}[E]\ h\ l\ m \rangle \\
\mathcal{I}[\langle E_1, E_2 \rangle \texttt{ in } R]\ h\ l\ m = m \cup \langle R, \langle \mathcal{E}[E_1]\ h\ l\ m, \mathcal{E}[E_2]\ h\ l\ m \rangle \rangle \\
\mathcal{E}[V]\ h\ l\ m = l(V) \\
\mathcal{E}[number]\ h\ l\ m = number \\
\mathcal{E}[E.field]\ h\ l\ m = b \text{ such that } \langle (\mathcal{E}[E]\ h\ l\ m), field, b \rangle \in h \\
\mathcal{E}[E_1.field[E_2]]\ h\ l\ m = \\
\qquad c \text{ such that } \langle (\mathcal{E}[E_1]\ h\ l\ m), field, (\mathcal{E}[E_2]\ h\ l\ m), c \rangle \in h \\
\mathcal{E}[E_1 \oplus E_2]\ h\ l\ m = primop(\oplus, (\mathcal{E}[E_1]\ h\ l\ m), (\mathcal{E}[E_2]\ h\ l\ m)) \\
\mathcal{E}[string]\ h\ l\ m = string
\end{array}
$$

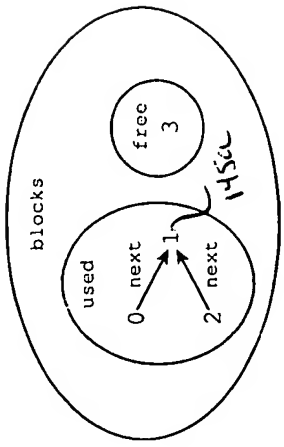
FIGURE 11 **Denotational Semantics for Model Definition Language**

TITLE: SPECIFICATION BASED DETECTION AND REPAIR OF ERRORS IN DATA STRUCTURES
Inventors: Brian C. Demsky, et al.
Filed: November 26, 2003 MIS-00401
Agent: Anne E. Saturnelli Reg. No.: 41,290 Sheet 11 of 28

```
Disk disk;

for i in 0..NumEntries, disk.table[i].valid &&
  disk.table[i].first < NumBlocks =>
  disk.table[i].first in used;
for b in used, 0 <= disk.FAT[b] &&
  disk.FAT[b] < NumBlocks => disk.FAT[b] in used;
for b in used, 0 <= disk.FAT[b] &&
  disk.FAT[b] < NumBlocks =>
  <b,disk.FAT[b]> in next;
for b in 0..NumBlocks, !(b in used) => b in free;
```

FIGURE 12A **Model Definition Declarations and Rules**



Inconsistent Model

144

136

```
C  := Q,C | B
Q  := for V in S | for V = E .. E
B  := B and B | B or B |!B | (B) |
      VE = E | VE < E | VE <= E | VE > E |
      VE >= E | V in SE | size(SE) = C |
      size(SE) >= C | size(SE) <= C
VE := V.R
E  := V | number | string | E + E | E - E | E/E |
      E * E | E.R | size(SE) | (E)
SE := S | V.R | R.V
```

FIGURE 13 **Internal Constraint Language**

138

$$
\begin{array}{lcl}
v & \in & Value = Number \cup Boolean \cup string \cup Object \\
l & \in & Local = \mathcal{P}(Var \times Value) \\
m & \in & Model = \mathcal{P}(Var \times Store) \\
s & \in & Store = Value \times Value \cup Value \\
\mathcal{EV} & : & C \rightarrow Local \rightarrow Model \rightarrow Boolean \\
\mathcal{E} & : & E \rightarrow Local \rightarrow Model \rightarrow Value \\
\mathcal{C} & : & B \rightarrow Local \rightarrow Model \rightarrow Boolean \\
\mathcal{V} & : & VE \rightarrow Local \rightarrow Model \rightarrow Value \\
\mathcal{SE} & : & SE \rightarrow Local \rightarrow Model \rightarrow \mathcal{P}(Value)
\end{array}
$$

$$
\begin{array}{l}
\mathcal{EV}[\texttt{for } V \texttt{ in } S, C]\ l\ m\ = \\
\quad \bigwedge_{v \in m(S)} \mathcal{EV}[C]\ l[V \mapsto v]\ m \\
\mathcal{EV}[\texttt{for } V = E_1..E_2, C]\ l\ m\ = \\
\quad \bigwedge_{v=\mathcal{E}[E_1] l m}^{\mathcal{E}[E_2] l m} \mathcal{EV}[C]\ l[V \mapsto v]\ m \\
\mathcal{EV}[B]\ l\ m = \mathcal{C}[B]\ l\ m \\
\mathcal{C}[!B]\ l\ m = \neg\mathcal{C}[B]\ l\ m \\
\mathcal{C}[B_1 \texttt{ and } B_2]\ l\ m = \mathcal{C}[B_1]\ l\ m \wedge \mathcal{C}[B_2]\ l\ m \\
\mathcal{C}[B_1 \texttt{ or } B_2]\ l\ m\ = \mathcal{C}[B_1]\ l\ m \vee \mathcal{C}[B_2]\ l\ m \\
\mathcal{C}[V \texttt{ in } SE]\ l\ m\ = l(V) \in \mathcal{SE}[SE]\ l\ m \\
\mathcal{C}[VE = E]\ l\ m\ = (\mathcal{V}[VE]\ l\ m == \mathcal{E}[E]\ l\ m) \\
\mathcal{C}[VE < E]\ l\ m\ = (\mathcal{V}[VE]\ l\ m < \mathcal{E}[E]\ l\ m) \\
\mathcal{C}[VE <= E]\ l\ m\ = (\mathcal{V}[VE]\ l\ m \leq \mathcal{E}[E]\ l\ m) \\
\mathcal{C}[VE > E]\ l\ m\ = (\mathcal{V}[VE]\ l\ m > \mathcal{E}[E]\ l\ m) \\
\mathcal{C}[VE >= E]\ l\ m\ = (\mathcal{V}[VE]\ l\ m \geq \mathcal{E}[E]\ l\ m) \\
\mathcal{C}[\texttt{size}(SE) = C]\ l\ m\ = \mathcal{E}[\texttt{size}(SE)]\ l\ m\ == C \\
\mathcal{C}[\texttt{size}(SE) >= C]\ l\ m\ = \mathcal{E}[\texttt{size}(SE)]\ l\ m\ \geq C \\
\mathcal{C}[\texttt{size}(SE) <= C]\ l\ m\ = \mathcal{E}[\texttt{size}(SE)]\ l\ m\ \leq C \\
\mathcal{V}[V.R]\ l\ m\ = y \text{ such that } \langle l(V), y\rangle \in m(R) \\
\mathcal{E}[\texttt{size}(SE)]\ l\ m\ = \mid \mathcal{SE}[SE]\ l\ m \mid \\
\mathcal{E}[V]\ l\ m\ = l(V) \\
\mathcal{E}[E.R]\ l\ m\ = y \text{ such that } \exists z, z \in \mathcal{E}[E]\ l\ m \wedge \langle z, y\rangle \in m(R) \\
\mathcal{E}[E_1 \oplus E_2]\ l\ m\ = primop(\oplus, \mathcal{E}[E_1]\ l\ m, \mathcal{E}[E_2]\ l\ m) \\
\mathcal{SE}[S]\ l\ m\ = \{s \mid s \in m(S)\} \\
\mathcal{SE}[V.R]\ l\ m\ = \{y \mid \langle l(V), y\rangle \in m(R)\} \\
\mathcal{SE}[R.V]\ l\ m\ = \{y \mid \langle y, l(V)\rangle \in m(R)\}
\end{array}
$$

FIGURE 14 **Denotational Semantics for Internal Constraint Language**

TITLE: **SPECIFICATION BASED DETECTION AND REPAIR OF ERRORS IN DATA STRUCTURES**
Inventors: Brian C. Demsky, et al.
Filed: November 26, 2003 MIS-00401
Agent: Anne E. Saturnelli Reg. No.: 41,290 **Sheet 16 of 28**

1400

```
R  := Q, R | G ⇒ C
Q  := for V in S | for ⟨V, V⟩ in R | for V = E .. E
G  := G and G | G or G | !G | E = E | E < E | true
C  := HE.field = E | HE.field[E] = E | V = E
HE := V | HE.field | HE.field[E]
E  := V | number | string | E.R | E − E | E + E |
      E ∗ E | E/E | size(SE) | element E of SE
SE := S | V.R | R.V
```

FIGURE 15 **External Constraint Language**

142

$$
\begin{array}{rcl}
hv & \in & HeapValue = Bit \cup Byte \cup Short \cup Integer \cup Struct \\
h & \in & Heap = \mathcal{P}(Object \times Field \times HeapValue \cup \\
 & & Object \times Field \times \mathbb{N} \times HeapValue) \\
v & \in & Value = \mathbb{Z} \cup Boolean \cup string \cup Struct \\
l & \in & Local = Var \rightharpoonup Value \\
s & \in & Store = Value \times Value \cup Value \\
m & \in & Model = (P)(Var \times Store) \\
\mathcal{R} & : & R \rightarrow Heap \rightarrow Local \rightarrow Model \rightarrow Boolean \\
\mathcal{E} & : & E \rightarrow Heap \rightarrow Local \rightarrow Model \rightarrow Value \\
\mathcal{HE} & : & HE \rightarrow Heap \rightarrow Local \rightarrow Model \rightarrow Object \\
\mathcal{G} & : & G \rightarrow Heap \rightarrow Local \rightarrow Model \rightarrow Boolean \\
\mathcal{C} & : & C \rightarrow Heap \rightarrow Local \rightarrow Model \rightarrow Boolean \\
\mathcal{SE} & : & SE \rightarrow Local \rightarrow Model \rightarrow Value
\end{array}
$$

$$
\begin{array}{l}
\mathcal{R}[\texttt{for } V \texttt{ in } S, R]h\ l\ m\ = \bigwedge_{v \in m(S)} \mathcal{R}[R]hl[V \mapsto v]m \\
\mathcal{R}[\texttt{for } \langle V_1, V_2\rangle \texttt{ in } R, R]\ h\ l\ m\ = \bigwedge_{(v_1,v_2)\in m(R)} \\
\quad \mathcal{R}[R]\ h\ l[V_1 \mapsto v_1][V_2 \mapsto v_2]\ m \\
\mathcal{R}[\texttt{for } V = E_1 \,..\, E_2, R]\ h\ l\ m\ = \bigwedge_{v=\mathcal{E}[E_1]\ h\ l\ m}^{\mathcal{E}[E_2]\ h\ l\ m} \\
\quad \mathcal{R}[R]\ h\ l[V \mapsto v]\ m \\
\mathcal{R}[G \Rightarrow C]\ h\ l\ m\ = (\neg\mathcal{G}[G]\ h\ l\ m) \vee \mathcal{C}[C]\ h\ l\ m) \\
\mathcal{G}[G_1 \texttt{ and } G_2]\ h\ l\ m\ = (\mathcal{G}[G_1]\ h\ l\ m) \wedge (\mathcal{G}[G_2]\ h\ l\ m) \\
\mathcal{G}[G_1 \texttt{ or } G_2]\ h\ l\ m\ = (\mathcal{G}[G_1]\ h\ l\ m) \vee (\mathcal{G}[G_2]\ h\ l\ m) \\
\mathcal{G}[!G_1]\ h\ l\ m\ = \neg(\mathcal{G}[G_1]\ h\ l\ m) \\
\mathcal{G}[E_1 = E_2]\ h\ l\ m\ = (\mathcal{E}[E_1]\ h\ l\ m) == (\mathcal{E}[E_2]\ h\ l\ m) \\
\mathcal{G}[E_1 < E_2]\ h\ l\ m = (\mathcal{E}[E_1]\ h\ l\ m) < (\mathcal{E}[E_2]\ h\ l\ m) \\
\mathcal{G}[\texttt{true}]\ h\ l\ m = true \\
\mathcal{C}[HE.field = E]\ h\ l\ m = \langle \mathcal{HE}[HE]\ h\ l\ m, field, \mathcal{E}[E]\ h\ l\ m\rangle \in h \\
\mathcal{C}[HE.field[E_1] = E_2]\ h\ l\ m = \\
\quad \langle \mathcal{HE}[HE]\ h\ l\ m, field, \mathcal{E}[E_1]\ h\ l\ m, \mathcal{E}[E_2]\ h\ l\ m\rangle \in h \\
\mathcal{C}[V = E]\ h\ l\ m = (l(V) == \mathcal{E}[E]\ hl\ m) \\
\mathcal{HE}[V]\ h\ l\ m\ = l(V) \\
\mathcal{HE}[HE.field]\ h\ l\ m\ = b \text{ such that } \langle \mathcal{HE}[HE]\ h\ l\ m, field, b\rangle \in h \\
\mathcal{HE}[HE.field[E]]\ h\ l\ m\ = \\
\quad b \text{ such that } \langle \mathcal{HE}[HE]\ h\ l\ m, field, \mathcal{E}[E]\ h\ l\ m, b\rangle \in h \\
\mathcal{E}[V]\ h\ l\ m = l(V) \\
\mathcal{E}[number]\ h\ l\ m = number \\
\mathcal{E}[V.R]\ h\ l\ m\ = b \text{ such that } \langle V, b\rangle \in m(R) \\
\mathcal{E}[E_1 \oplus E_2]\ h\ l\ m = primop(\oplus, (\mathcal{E}[E_1]\ h\ l\ m), (\mathcal{E}[E_2]\ h\ l\ m)) \\
\mathcal{E}[string]\ h\ l\ m = string \\
\mathcal{E}[\texttt{size}(SE)]\ h\ l\ m = \mid \mathcal{SE}[SE]\ l\ m \mid \\
\mathcal{E}[\texttt{element } E \texttt{ of } SE]\ h\ l\ m = \text{given some ordering of } \mathcal{SE}[SE]\ l\ m, \\
\quad \text{pick element number } \mathcal{E}[E]\ h\ l\ m \\
\mathcal{SE}[S]\ l\ m = \{s \mid s \in m(S)\} \\
\mathcal{SE}[V.R]\ !\ m\ = \{y \mid \langle l(V), y\rangle \in R\} \\
\mathcal{SE}[R.V]\ l\ m\ = \{y \mid \langle y, l(V)\rangle \in R\}
\end{array}
$$

FIGURE 16

Denotational Semantics for External Constraint Language

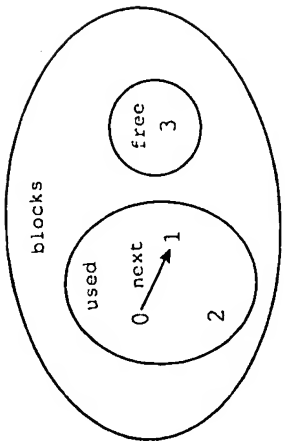


Figure 17: Repaired Model

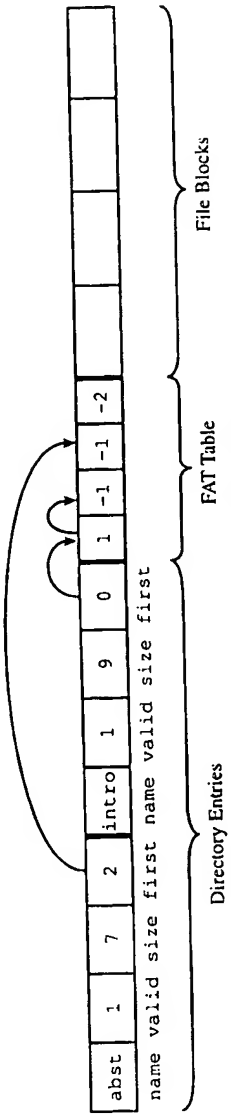


Figure 18 Repaired File System

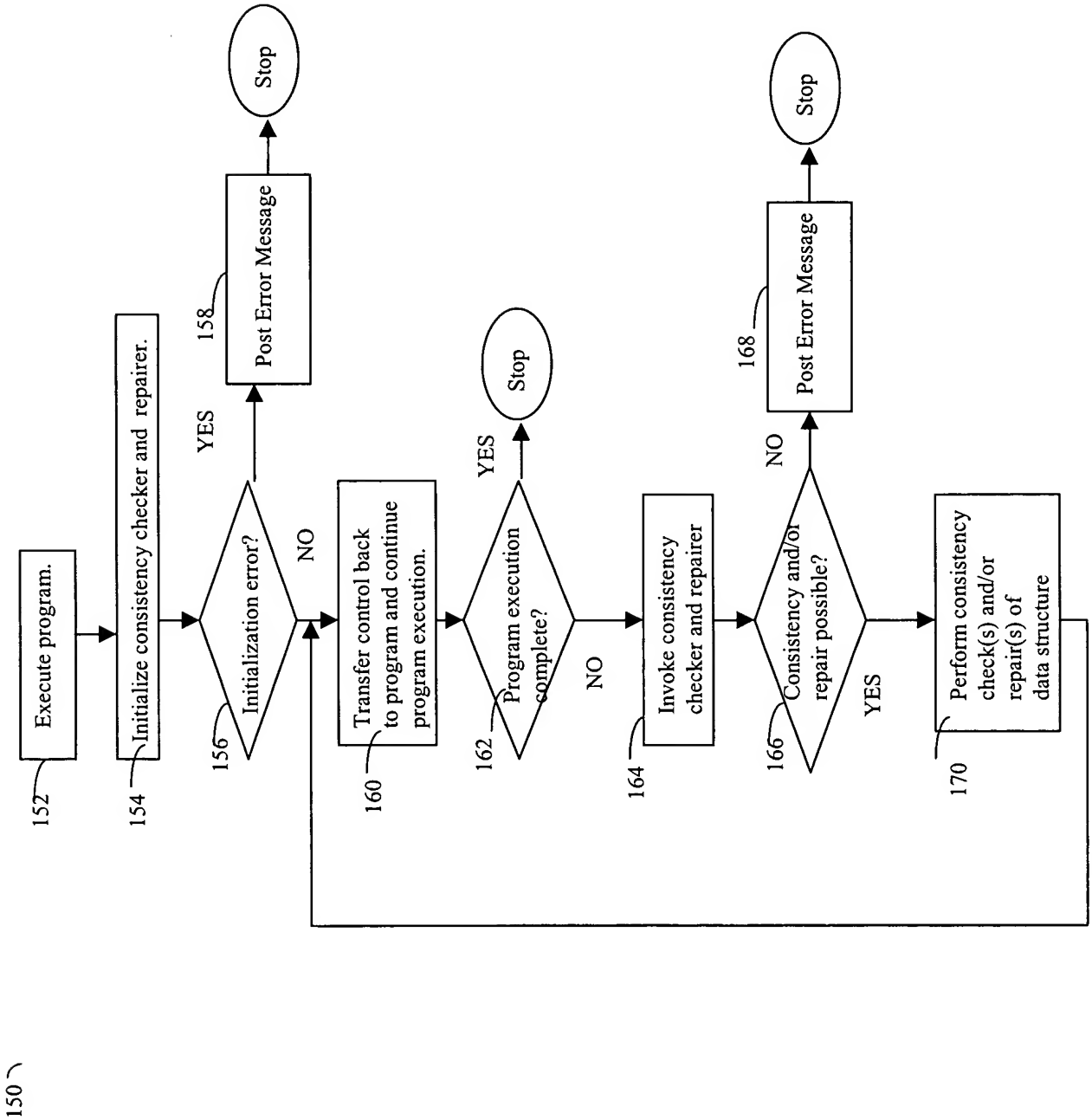

150
Execute program.
152
Initialize consistency checker and repairer.
154
Initialization error?
156
YES
Post Error Message
158
Stop
NO
Transfer control back to program and continue program execution.
160
Program execution complete?
162
YES
Stop
NO
Invoke consistency checker and repairer
164
Consistency and/or repair possible?
166
NO
Post Error Message
168
Stop
YES
Perform consistency check(s) and/or repair(s) of data structure
170

FIGURE 19

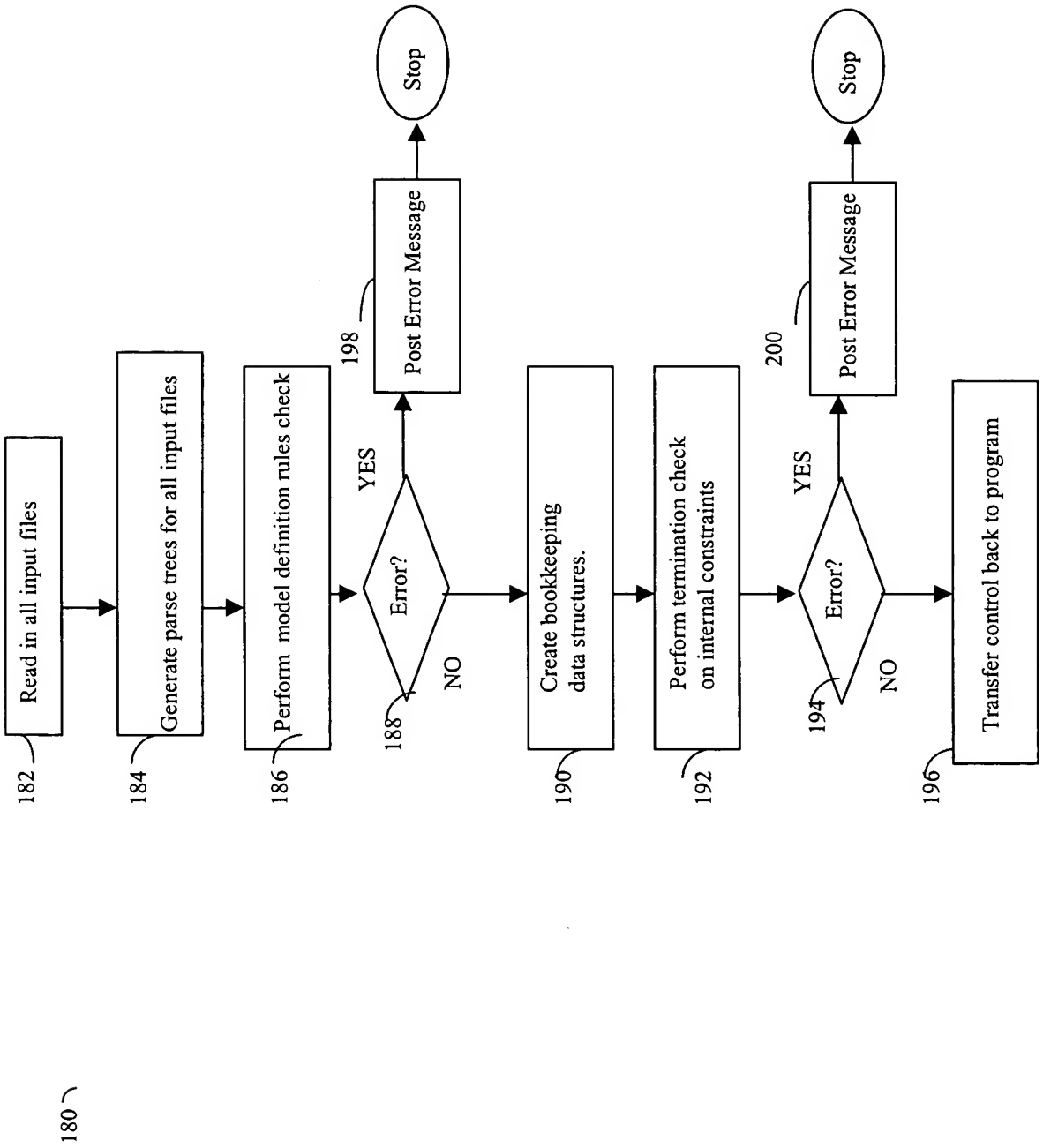

180
Read in all input files
182
Generate parse trees for all input files
184
Perform model definition rules check
186
Error?
188
YES
NO
Post Error Message
198
Stop
Create bookkeeping data structures.
190
Perform termination check on internal constraints
192
Error?
194
YES
NO
Post Error Message
200
Stop
Transfer control back to program
196

FIGURE 20

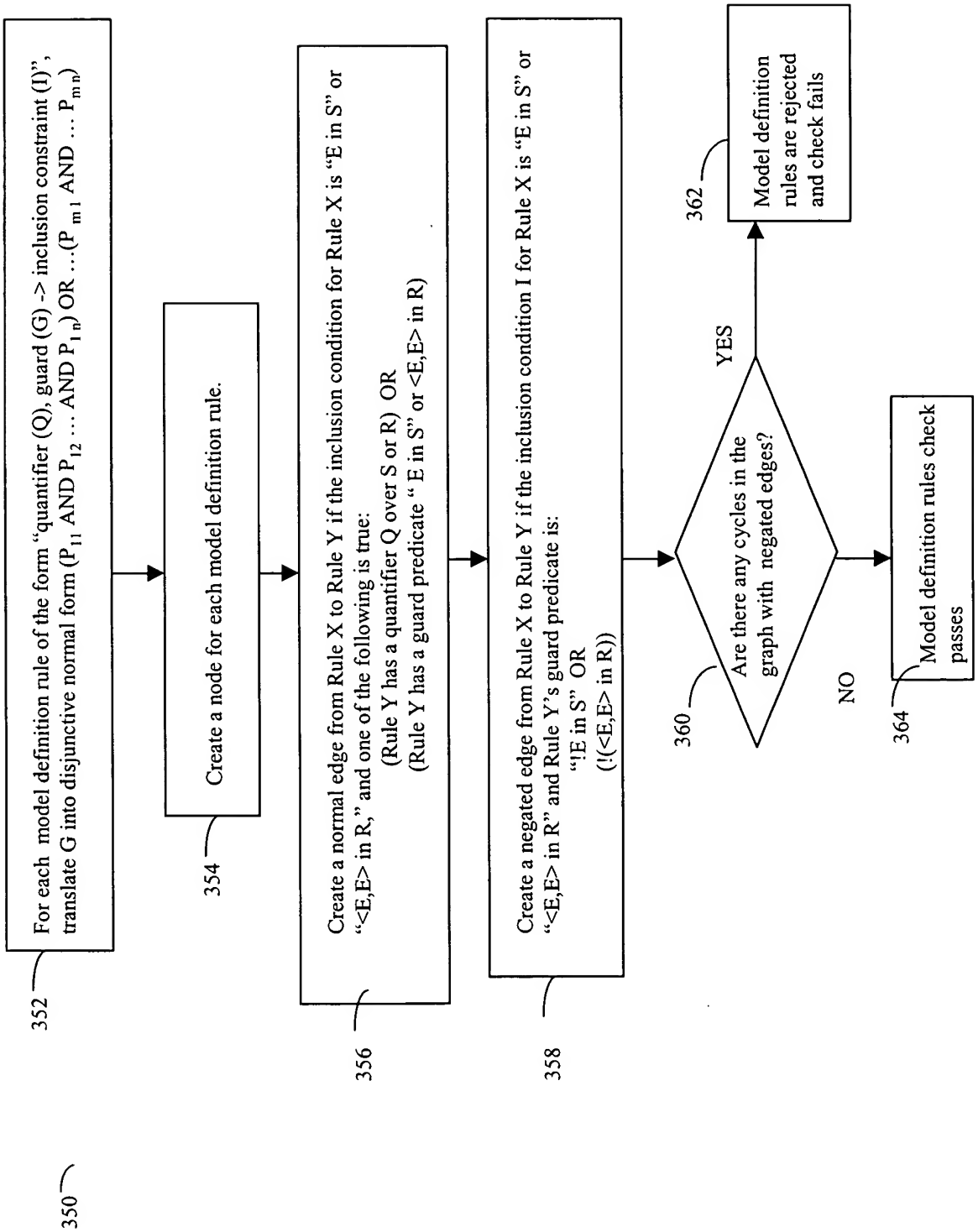

350
352
For each model definition rule of the form "quantifier (Q), guard (G) -> inclusion constraint (I)", translate G into disjunctive normal form (P_{11} AND P_{12} ... AND P_{1n}) OR ...(P_{m1} AND ... P_{mn})
354
Create a node for each model definition rule.
356
Create a normal edge from Rule X to Rule Y if the inclusion condition for Rule X is "E in S" or "<E,E> in R," and one of the following is true:
(Rule Y has a quantifier Q over S or R) OR
(Rule Y has a guard predicate " E in S" or <E,E> in R)
358
Create a negated edge from Rule X to Rule Y if the inclusion condition I for Rule X is "E in S" or "<E,E> in R" and Rule Y's guard predicate is:
"!E in S" OR
(!(<E,E> in R))
360
Are there any cycles in the graph with negated edges?
YES
362
Model definition rules are rejected and check fails
NO
364
Model definition rules check passes

FIGURE 21

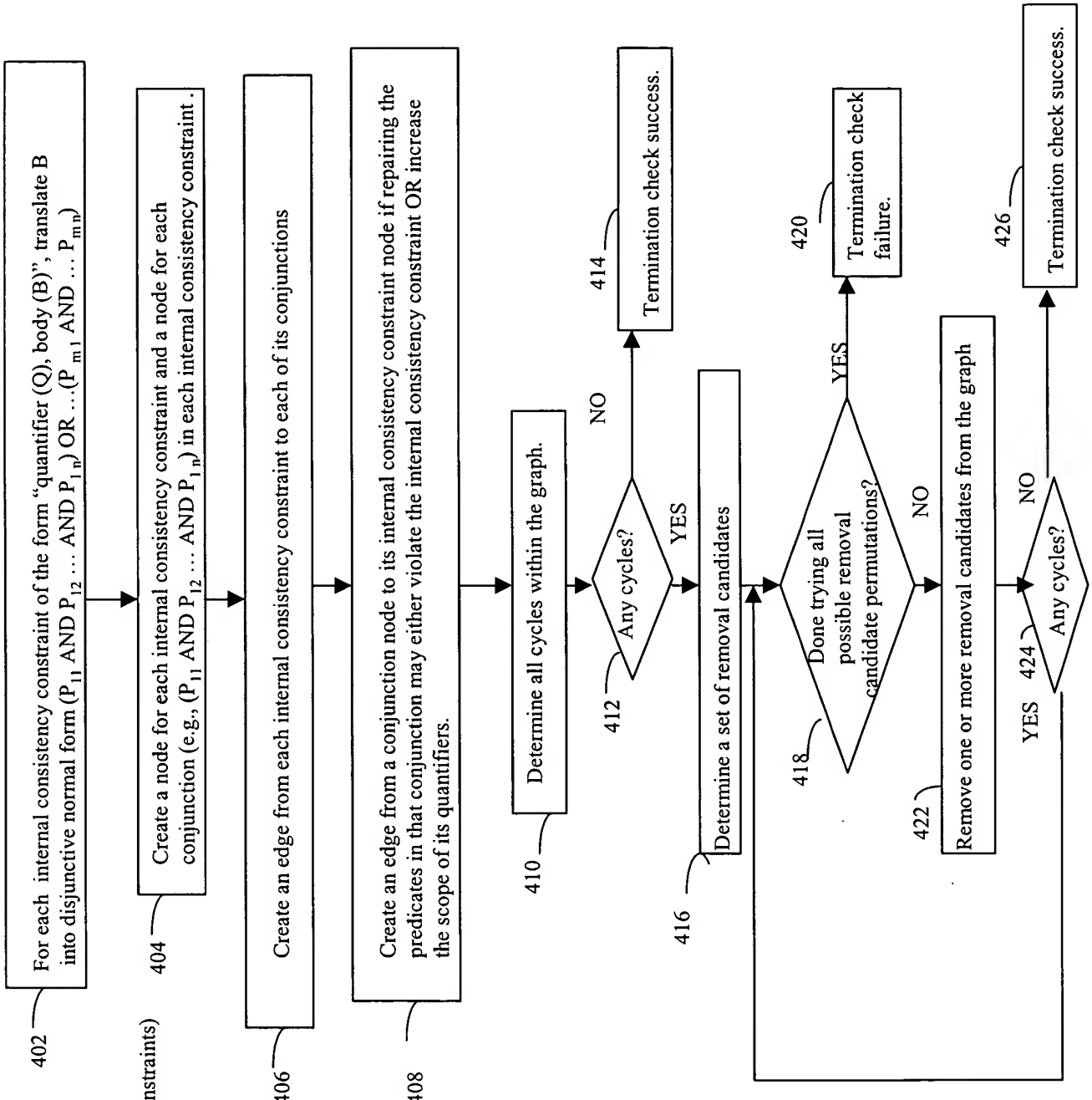

400
(termination check on internal consistency constraints)
402
For each internal consistency constraint of the form "quantifier (Q), body (B)", translate B into disjunctive normal form (P_{11} AND P_{12} ... AND P_{1n}) OR ...(P_{m1} AND ... P_{mn})
404
Create a node for each internal consistency constraint and a node for each conjunction (e.g., (P_{11} AND P_{12} ... AND P_{1n}) in each internal consistency constraint .
406
Create an edge from each internal consistency constraint to each of its conjunctions
408
Create an edge from a conjunction node to its internal consistency constraint node if repairing the predicates in that conjunction may either violate the internal consistency constraint OR increase the scope of its quantifiers.
410
Determine all cycles within the graph.
412
Any cycles?
NO
414
Termination check success.
YES
416
Determine a set of removal candidates
418
Done trying all possible removal candidate permutations?
YES
420
Termination check failure.
NO
422
Remove one or more removal candidates from the graph
424
Any cycles?
NO
426
Termination check success.
YES

FIGURE 22

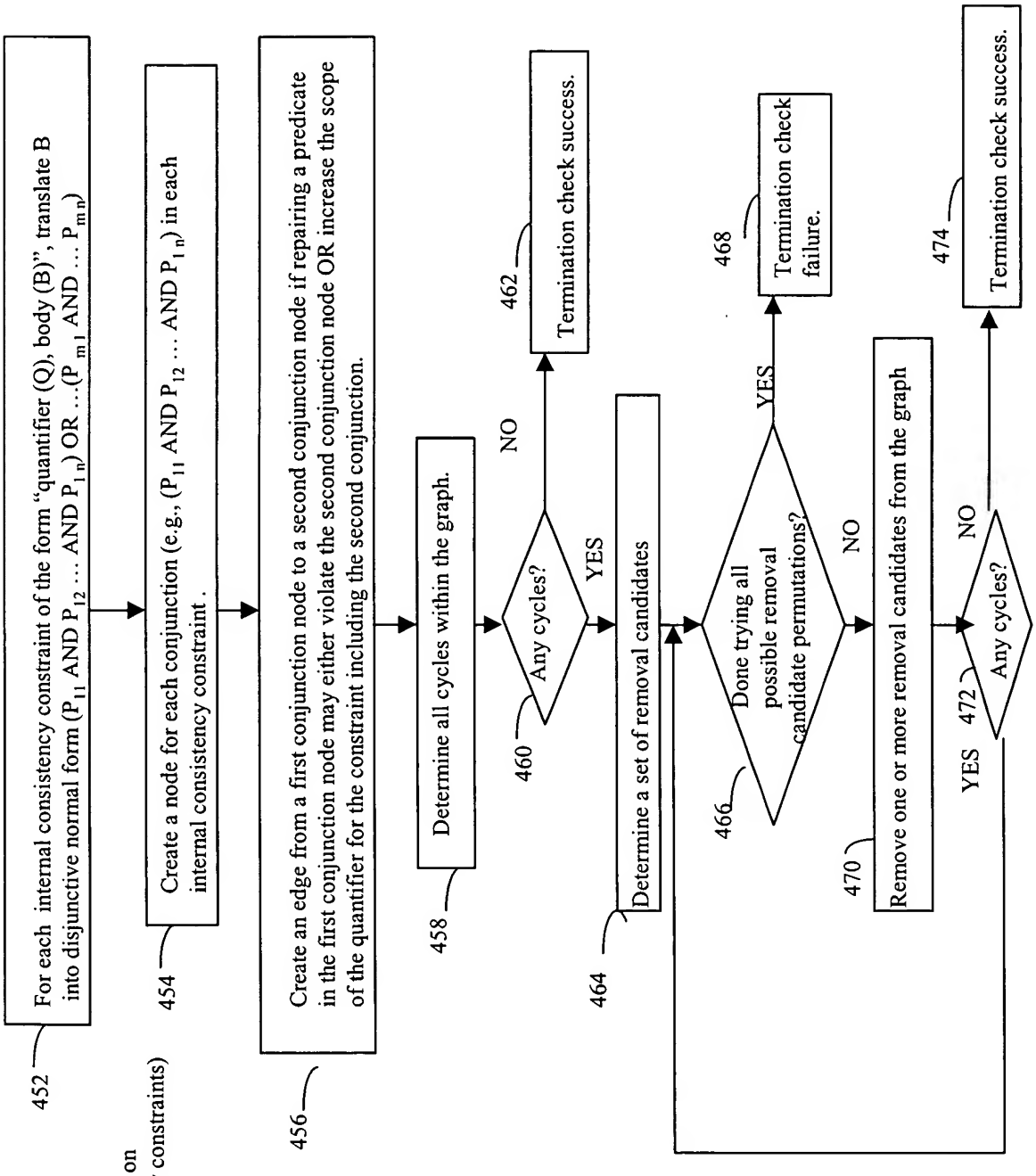

450
(termination check on internal consistency constraints)
452
For each internal consistency constraint of the form "quantifier (Q), body (B)", translate B into disjunctive normal form (P_{11} AND P_{12} ... AND P_{1n}) OR ...(P_{m1} AND ... P_{mn})
454
Create a node for each conjunction (e.g., (P_{11} AND P_{12} ... AND P_{1n}) in each internal consistency constraint.
456
Create an edge from a first conjunction node to a second conjunction node if repairing a predicate in the first conjunction node may either violate the second conjunction node OR increase the scope of the quantifier for the constraint including the second conjunction.
458
Determine all cycles within the graph.
460
Any cycles?
NO
462
Termination check success.
YES
464
Determine a set of removal candidates
466
Done trying all possible removal candidate permutations?
YES
468
Termination check failure.
NO
470
Remove one or more removal candidates from the graph
472
Any cycles?
NO
474
Termination check success.
YES

FIGURE 23

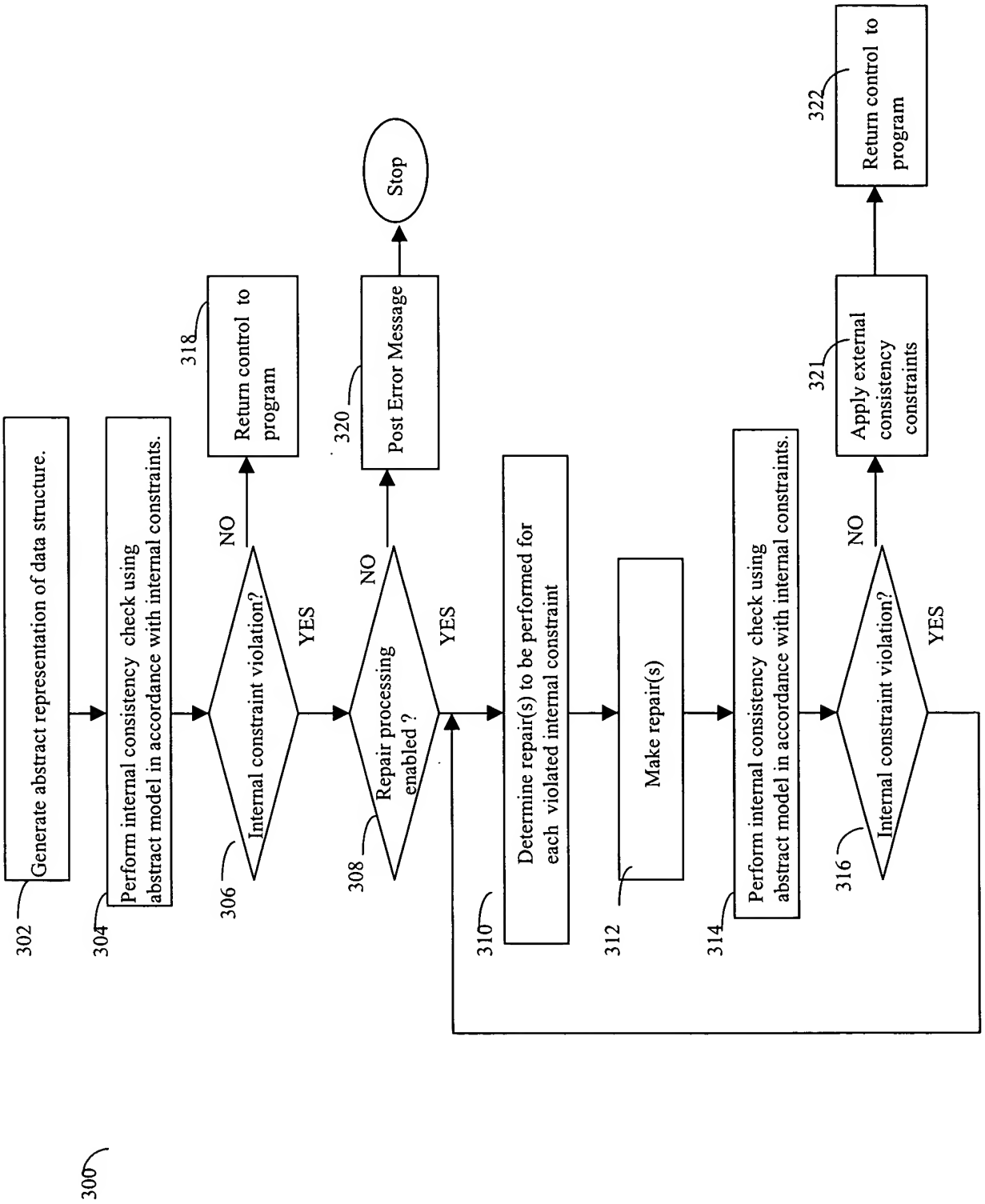

300
302
Generate abstract representation of data structure.
304
Perform internal consistency check using abstract model in accordance with internal constraints.
306
Internal constraint violation?
NO
318
Return control to program
YES
308
Repair processing enabled ?
NO
320
Post Error Message
Stop
YES
310
Determine repair(s) to be performed for each violated internal constraint
312
Make repair(s)
314
Perform internal consistency check using abstract model in accordance with internal constraints.
316
Internal constraint violation?
NO
321
Apply external consistency constraints
322
Return control to program
YES

FIGURE 24

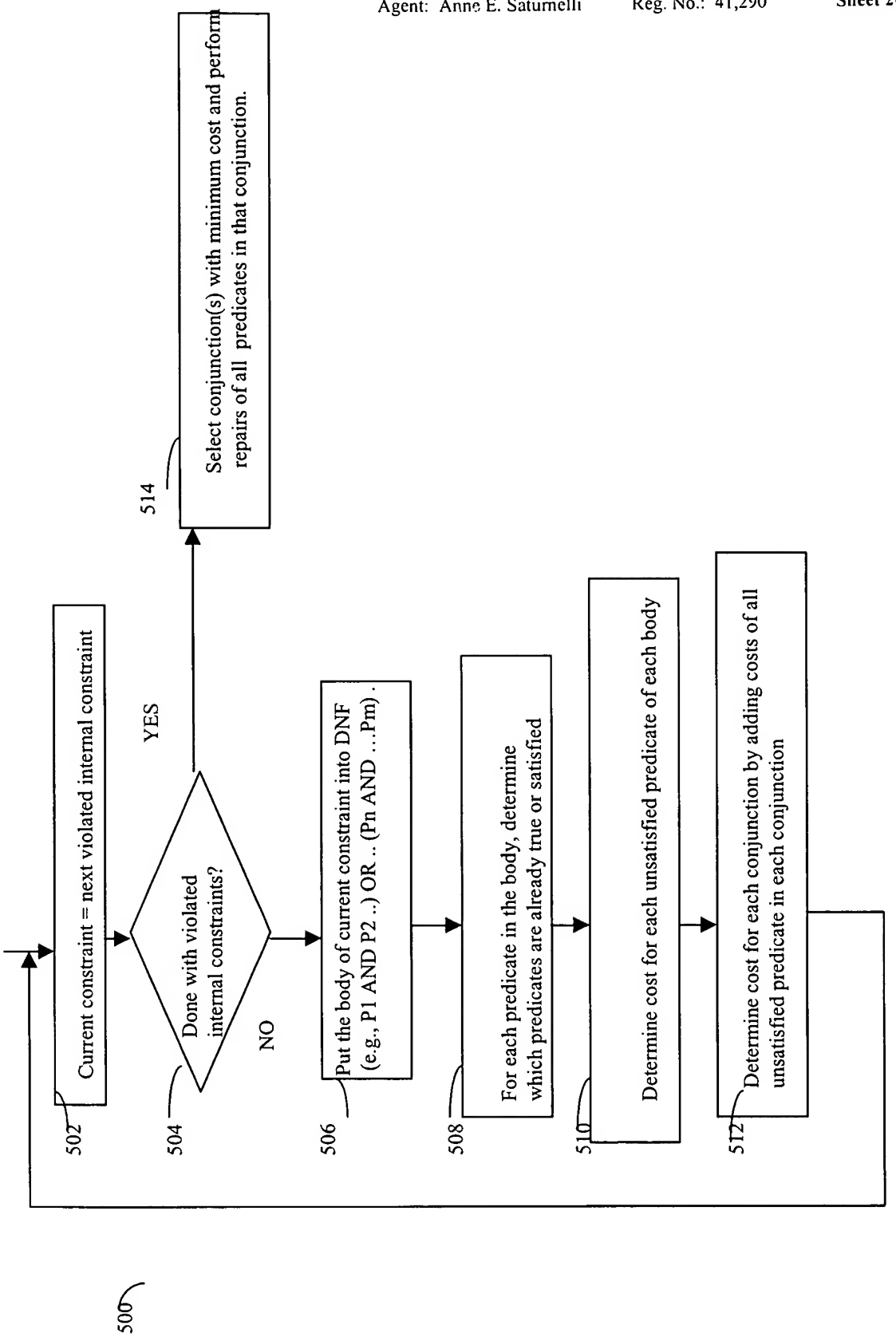

500
502
Current constraint = next violated internal constraint
504
Done with violated internal constraints?
YES
NO
506
Put the body of current constraint into DNF (e.g., P1 AND P2 ..) OR .. (Pn AND ...Pm) .
508
For each predicate in the body, determine which predicates are already true or satisfied
510
Determine cost for each unsatisfied predicate of each body
512
Determine cost for each conjunction by adding costs of all unsatisfied predicate in each conjunction
514
Select conjunction(s) with minimum cost and perform repairs of all predicates in that conjunction.

FIGURE 25

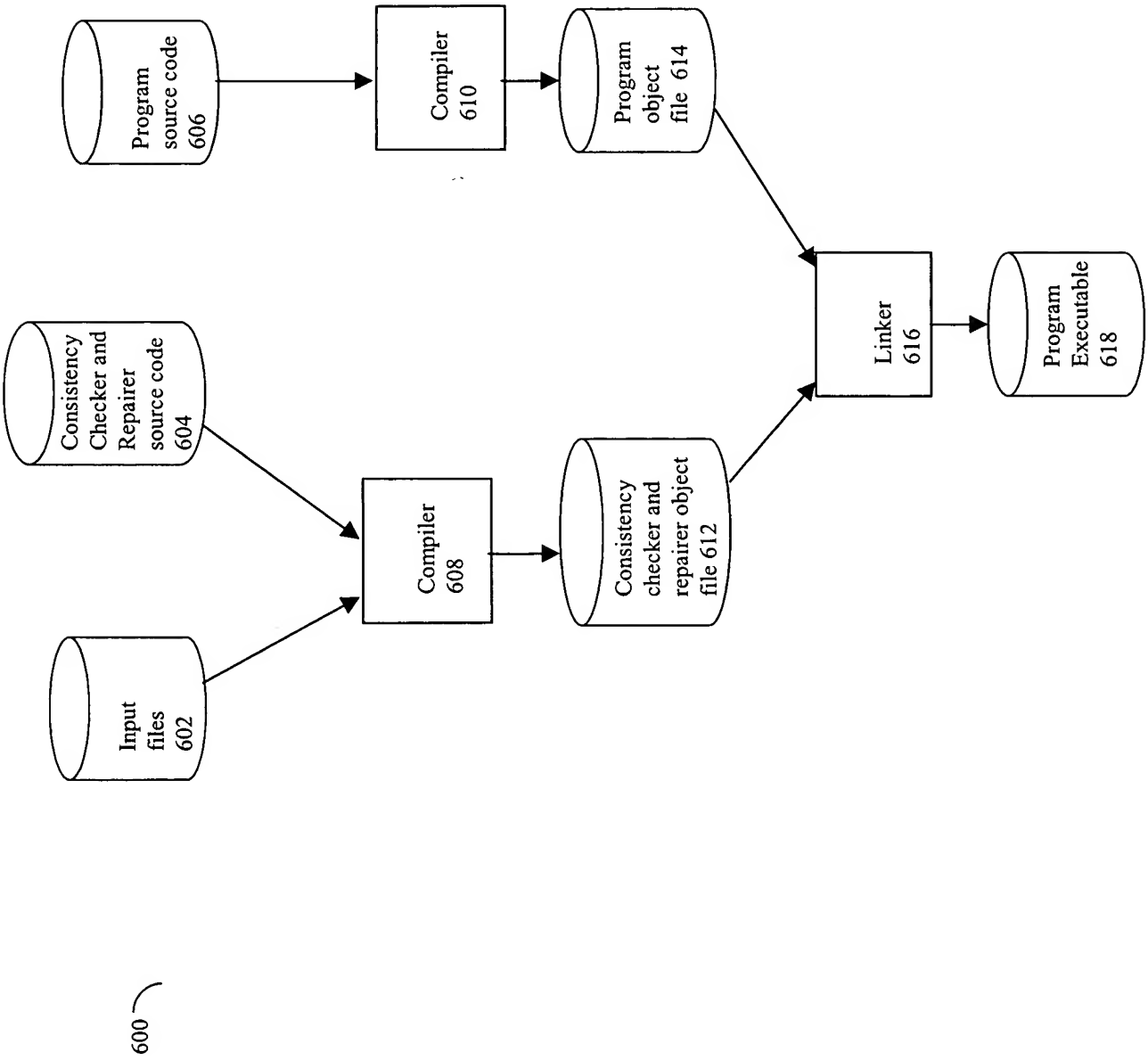

Program source code 606
Compiler 610
Program object file 614
Consistency Checker and Repairer source code 604
Linker 616
Program Executable 618
Compiler 608
Consistency checker and repairer object file 612
Input files 602
600

FIGURE 26

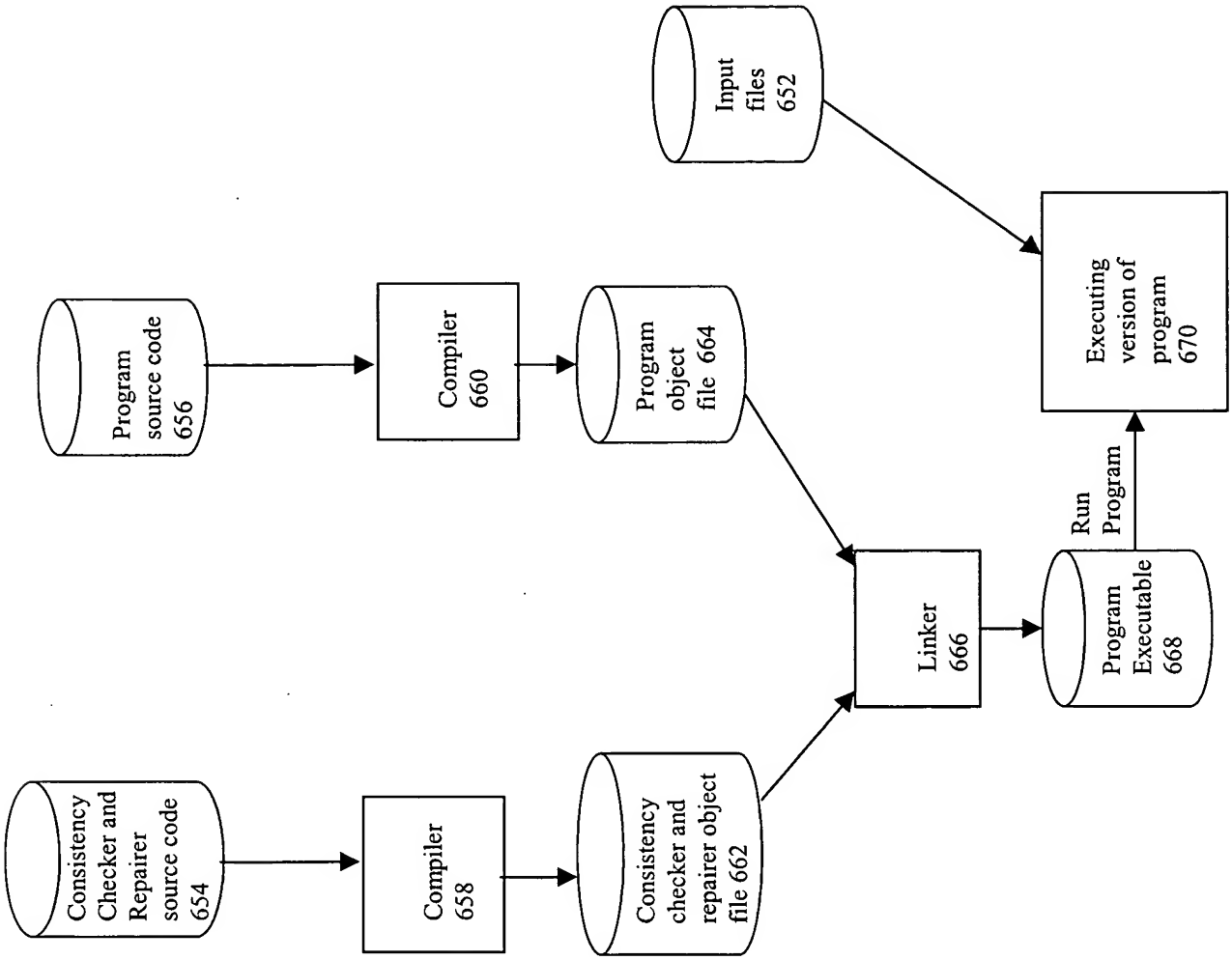

Input files 652
Executing version of program 670
Program source code 656
Compiler 660
Program object file 664
Run Program
Linker 666
Program Executable 668
Consistency Checker and Repairer source code 654
Compiler 658
Consistency checker and repairer object file 662
650

FIGURE 27